Binary and multiclass classification on MNIST dataset

Rishabh Bhattacharya
Department of Mechanical and Aerospace Engineering
University of California, San Diego
La Jolla, CA 92092
ribhattacharya@ucsd.edu

Shrey Kansal
Department of Mechanical and Aerospace Engineering
University of California, San Diego
La Jolla, CA 92092
skansal@ucsd.edu

Abstract

Given the MSNIST dataset, consisting of gray-scale images of handwritten digits from 0 through 9, we trained our Logistic Regression and Softmax Regression models in order to classify the digits. It was noticed that classification accuracy for digits 2 and 7 at 98.25% was better than that for digits 5 and 8 at 95.66%. Furthermore, the Softmax Regression classification accuracy for all 10 digits was lower at 92.37%.

1 Introduction

The MNIST data comprises of gray-scale images of handwritten digits ranging from 0 through 9. In order to implement two models of logistic regression (2v7 and 5v8), and one softmax regression for all 10 digits, the data was segregated accordingly. All training was done post reducing the feature dimensions through Pricnipal Component Analysis (PCA). Overfitting was reduced with the help of k-folds cross-validation and early stopping measures.

Hyperparameters including batch size, learning rate and number of PCA components were tuned through grid search. Softmax Regression model reported the lowest accuracy 92.37%, although greater than 90%, whereas Logistic Regression reported 98.25% and 95.66% for both models respectively. Upon visualizing the top 10 PCA components as images from Logistic Regression, it becomes apparent that the features are nothing but shapes of the digits. However, the same is not exactly apparent in case of Softmax Regression.

2 Related Work

We referred to Prof. Gary Cottrell's lecture notes (1, 1A, 2 and 2A) from the CSE 251B (Winter 23) resources.

3 Dataset

The training dataset consists of 60k samples in the form of 28x28 gray-scale images, whereas the test data is comprised of 10k samples in similar format. The handwritten digits range from 0 through

9, i.e., 10 classes. Each class contributes approximately 6k samples, with digit-1 class having the maximum samples (6742) and digit-5 class the minimum (5421). Among the test data examples as well, digit-5 and digit-1 classes have the minimum and maximum number of samples respectively.

A random image of digit-7 class can be visualized in Fig. 1. The images are normalized using the min-max normalization method, such that all pixel intensity values range between 0 and 1.

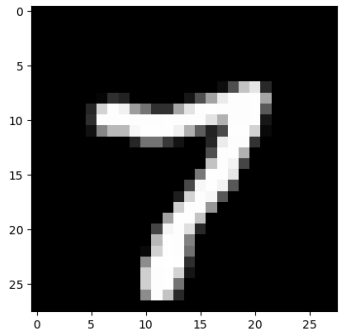

Figure 1: A random training image

In case of softmax regression, the data labels are converted to 1D vectors with one-hot encoding, such that the i-th class entry is 1, and the rest are 0. For instance, a one-hot encoded data label for digit-4 class can be visualized in Fig. 2.

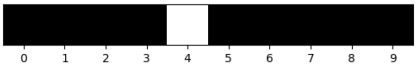

Figure 2: One-hot encoded label vector for digit-4

4 Principal Component Analysis (PCA)

PCA is a dimensionality-reduction method, where the original dataset is projected onto a p dimensional orthogonal space. If the original data is d dimensional, it is converted to be p dimensional using the projection matrix/principal components $\mathcal{U} \in \mathbb{R}^{p \times d}$. The orthonormal vectors/principal components (rows) in $\mathcal{U}$ are arranged in descending order of importance i.e. the top rows maintain most of the variance in our dataset. Since we have 10 classes (digits 0-9) in our training data, the top 10 principal components should account for most of the variance captured in our training.

$$\tilde{X}_{N \times p} = X_{N \times d} \, \mathcal{U}_{p \times d}^{\top} \tag{1}$$

where $\tilde{X}_{N \times p}$ is the reduced dimension dataset and $X_{N \times d}$ is the original dataset.

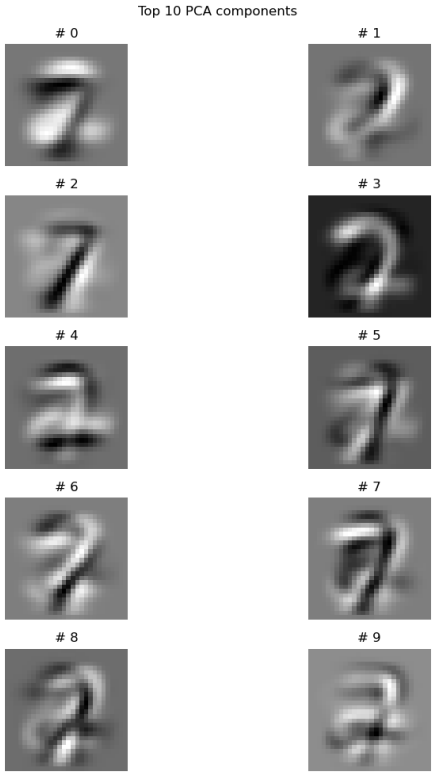

Figure 3: Top 10 PCA components for Linear regression - 2v7 model

From Figure 3, we can see that the top 10 PCA components are images of 2 and 7, which is because that is our dataset for the model. We see that these 10 components cover most of the variations of the digits, and thus are the most important components of the reduced dimensional data.

Similar to the previous plot, Figure 4 shows the top 10 components capturing most of the variation of the 5v8 model.

In Figure 5, we see that the top 10 components capture most of the digit variations. The components look like the digits in our dataset.

5 Logistic Regression

Logistic Regression (LR) is primarily used for binary classification, i.e., identifying a given d-dimensional input vector $x \in \mathbb{R}^d$ between two labels (0 and 1). In this project, we have implemented LR to classify the digit pairs (2,7) and (5,8). We have used Sigmoid function (2) as the logistic activation function, which gives us the conditional probability of the input data being in one of the target categories. The logistic unit is parameterized by a weight $w \in \mathbb{R}^{d+1}$, where w_0 represents the bias term.

$$y = P(\mathcal{C}_1|x) = \frac{1}{1 + \exp(-w^\top x)} = g(w^\top x) \tag{2}$$

$$P(\mathcal{C}_0|x) = 1 - P(\mathcal{C}_1|x) = 1 - y \tag{3}$$

where x has been augmented with a leading 1 to represent the bias input. To compute the error, we define the binary Cross-Entropy cost function, 4, which is the quantity we want to minimize over our training examples.

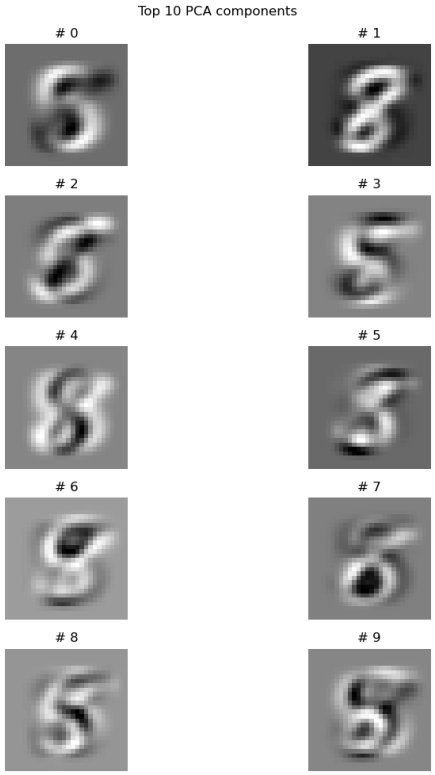

Figure 4: Top 10 PCA components for Linear regression - 5v8 model

$$E(w) = -\sum_{n=1}^{N} \{t^n \ln(y^n) + (1 - t^n) \ln(1 - y^n)\} \tag{4}$$

To minimize the cost function, we take its derivative, 5, and set it to 0.

$$-\frac{\partial E(w)}{\partial w_j} = \sum_{n=1}^{N} (t^n - y^n) x_j^n \tag{5}$$

5.1 Hyperparameter tuning

Hyperparameters: batch size, learning rate and number of PCA components were tuned using grid search based on the results from validation sets.

We selected 200 components based on the observation that most digits would fit inside a square of 14 pixels, which results in an area of 196 pixels. Reducing the number of PCA components would lower the validation accuracy as we are trying to fit a 784 dimensional vector into a lower dimensional vector. This results in huge amount of information loss, leading to inaccurate results. Increasing the number of components past 200 had negligible effect.

Learning rate was carefully tuned to avoid overfitting and reduce the noise in the results too. Upon further reducing the learning rate, we observed a smoother loss plot; however, overfitted on the training set. In order to avoid this, an iterative process was carried out for both models to get best possible learning rates. The said factor is considerably lower in case of 5v8 classification, as the two digits alike and are hard to classify,

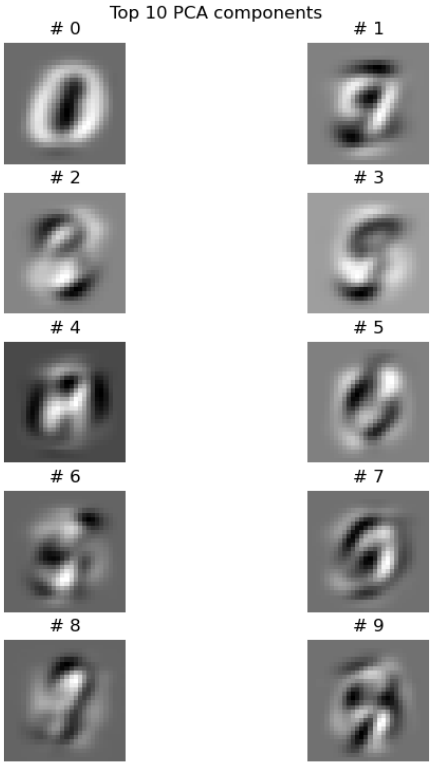

Figure 5: Top 10 PCA components for Softmax regression

The batch size too was tuned separately for both models through an iterative process. Its effect was prominent in case of 5v8 model as compared to 2v7; hence, we had to choose a lower batch size for the former.

5.2 Results

To prevent overfitting, we have implemented k-folds cross-validation algorithm, which takes $k-1$ mutually exclusive sets from the training data to train, and sets aside the remaining mutex subset for validation. The training procedure is run for M epochs on the k combinations of training and validations sets for hyperparameter tuning. The training and validation loss curves are plotted in Figures 6 and 7.

As we see the loss on validation set stops improving after $m = 15$ epochs for 2v7 model and $m = 14$ epochs for 5v8 model, an early stopping criteria is set to stop further training of the model. As expected, the validation loss is greater than training loss while maintaining the same curve shape. We take the weights trained upto m epochs to compute the accuracy of our test set, which is tabulated in Table 1.

Table 1: Logistic Regression - Test Set Accuracy

Model	**Accuracy**
2v7	98.25%
5v8	95.66%

As evident, the test accuracy in case of 2v7 classification is better than 5v8. This difference can be attributed to the fact that digits 5 and 8 are more alike each other in shape than digits 2 and 7.

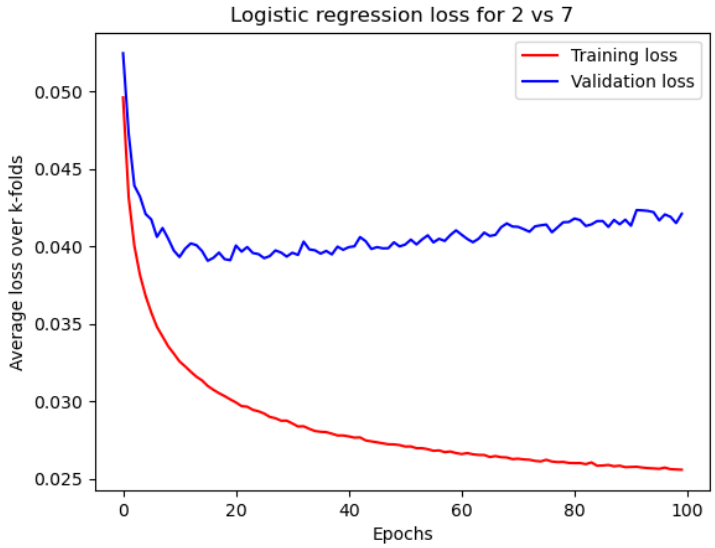

Figure 6: Loss over all epochs (averaged over k-folds)

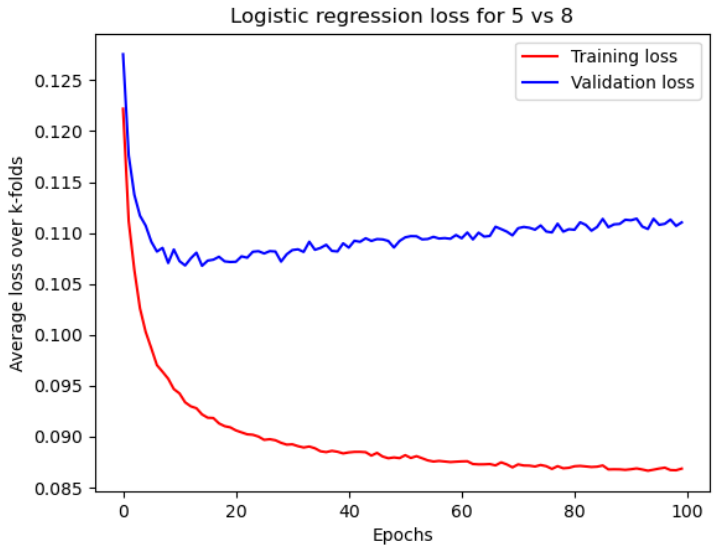

Figure 7: Loss over all epochs (averaged over k-folds)

Moreover, digit 5 has the least amount of training data (5421 examples), whereas digits 2 and 7 have more examples to train on.

The best hyperparameters selected for each Logistic Regression model is reported in Tables 2 and 3.

6 Softmax Regression

Softmax regression (SR) is the generalization of logistic regression for multiple (c) classes. Given an input $x^n \in \mathbb{R}^d$, softmax regression will output a vector $y^n \in \mathbb{R}^c$, where each element, y_k^n represents

Table 2: Logistic Regression - Hyperparameters for 2v7 Model

Hyperparameters	Values
Batch Size	50
Learning Rate	0.009
PCA Components	200

Table 3: Logistic Regression - Hyperparameters for 5v8 Model

Hyperparameters	Values
Batch Size	25
Learning Rate	0.006
PCA Components	200

the probability that x^n is in class k ($k \in [0, 1 \cdots, c-1]$. In our case, $c = 10$ since we want to classify new data into one of 10 classes (digits 0-9).

$$y_k^n = \frac{\exp\left(a_k^n\right)}{\sum_{k'} \exp\left(a_{k'}^n\right)} \tag{6}$$

where $a_k^n = w_k^T x^n \in \mathbb{R}$ is called the net input to output unit y_k. Equation 6 is called the softmax activation function, and it is a generalization of the logistic activation function. For softmax regression, we use a one hot encoding of the targets. That is, the targets are a c-dimensional vector, where the k^{th} element for example n (written t_k^n) is 1 if the input is from category k, and 0 otherwise. Note each output has its own weight vector w_k. With our model defined, we now define the cross-entropy cost function for multiple categories in Equation 7:

$$E = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{7}$$

which is what we want to minimize over our iterations (using gradient descent). Surprisingly, the gradient rule (Equation 8) turns out to be the same as that for logistic regression,

$$-\frac{\partial E(w)}{\partial w_{jk}} = \sum_{n=1}^{N} \left(t_k^n - y_k^n\right) x_j^n \tag{8}$$

where w_{jk} is the weight from the j^{th} input to the k^{th} output. Note that in softmax regression, we have a weight matrix $\mathbf{w} \in \mathbb{R}^{p \times c}$ (unlike a weight vector $\mathbb{R}^p$ in linear regression). Here p is the reduced dimension after performing PCA on the training dataset and $c = 10$ is the number of classes.

6.1 Hyperparameter tuning

Table 4: Softmax Regression - Hyperparameters

Hyperparameters	Values
Batch Size	50
Learning Rate	0.007
PCA Components	200

Table 5: Grid search for softmax regression

Hyperparameters	Values
Batch Size	25, 50, 100, 200, 500
Learning Rate	0.001, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01
PCA Components	30, 50, 100, 200, 500, 784

We performed grid search (Table 5)and tuned the hyperparameters as given in Table 4. We observed that we needed especially high PCA components for this regression, since the dimensionality reduction would need to maintain the variance of the entire dataset, instead of a subset (like in logistic regression). The batch size did not have a lot of effect in this model, and was thus kept similar to previous experiments. The validation/training loss plots (8) are very sensitive to the learning rate. A high learning rate would lead to a *jagged* graph (since it would bounce around the minima and never really converge). Also the trends (like *overfitting*) could not be observed. A low learning rate would take a long time to converge, leading to more computational resources.

6.2 Results

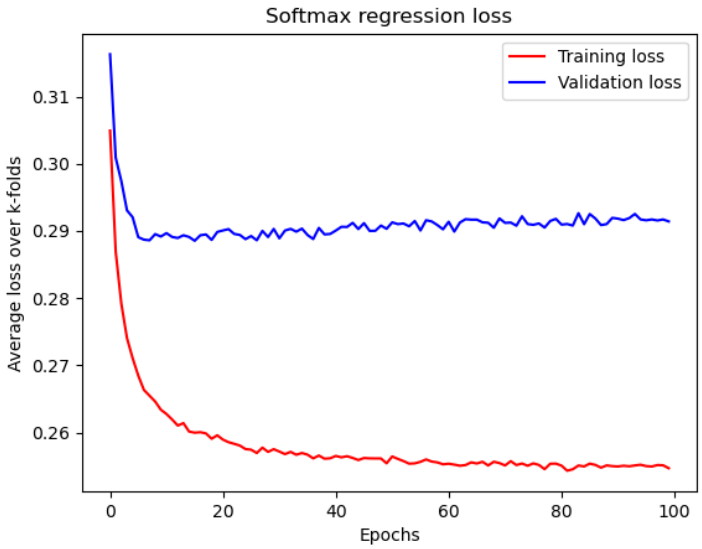

Figure 8: Loss over all epochs (averaged over k-folds)

From Figure 8, we observe that as expected, the training loss decreases with number of epochs. However, the validation loss reaches a minima and then starts rising again at epoch #15. This trend is very slight, and not very prevalent, like in the logistic regression cases. Albeit, this means that the model starts to overfit the data at around epoch #15, and thus needs to be *early stopped* (and its weights frozen). The validation loss is also greater than the training loss, which is expected since that is *unseen* data for the network.

Table 6: Softmax Regression - Test Set Accuracy

Model	Accuracy
Multi-class classifier	92.37%

We get an accuracy of 92.37% for the test set, which is quite good. It is lower than the accuracy of logistic regression, which is expected since this is a multi-class classifier (unlike binary classification in LR).

For $p = 30$ and $\delta = 0.005$ (pca dimensions and learning rate respectively), and all other hyper-paramters unchanged, we get an accuracy of 89.71%, with the minimum loss obtained at epoch #19.

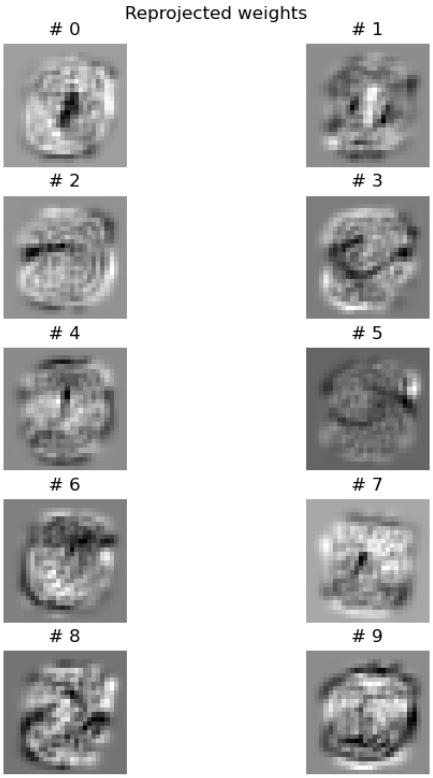

Figure 9: Weights visualized as 28×28 images

From Figure 9, we can see that the weights of the neural network approximately look like the digits they classify. While some digits are clearly visible (like 0), other digits are not as clear. This might be because digit 0 is very different from the rest, and is thus differentiated easily. Complicated digits like $2, 7$ and $5, 8$ are thus not as clear. Also, the neural network is being trained on a reduced dataset (PCA). This might lead to *reprojection errors*, due to which the weights are not as clear.

Team contributions

- Rishabh:
  - main.py, network.py, visual_utils.py, data.py
  - PCA, Softmax Regression
- Shrey:
  - main.py, network.py, visual_utils.py, data.py
  - Introduction, Related work, Dataset, Logistic regression